Class modifiers

package myPackage

public class

```
package myPackage;

public class PublicClass {
   …
}
```

package class

```
package myPackage;

class PackageClass {
   …
}
```

access from same package class

```
package myPackage;

public class SamePackageClass {
   PublicClass a = new PublicClass();
   PackageClass b = new PackageClass();
}
```

package otherPackage

access from other package class

```
import myPackage.*;

public class OtherPackageClass {
   PublicClass a = new PublicClass();
   PackageClass b = new PackageClass(); ✗
}
```

access to →	public class	default (package) class
same package class	✓	✓
other package class	✓	✗

G.V. Jourdan
gvj@eecs.uottawa.ca
http://www.eecs.uottawa.ca
Version 1.0

members and methods modifiers

package myPackage

same class access

```
package myPackage;

public class TestMemberAccessFromClass {

   public int publicMember;
   protected int protectedMember;
   int packageMember;
   private int privateMember;

   public TestMemberAccessFromClass() {
      publicMember = 0;
      protectedMember = 0;
      packageMember = 0;
      privateMember = 0;
   }
}
```

same package access

```
package myPackage;

public class TestMemberAccessFromPackage {

   TestMemberAccessFromClass test =
      new TestMemberAccessFromClass();

   public TestMemberAccessFromPackage() {
      test.publicMember = 0;
      test.protectedMember = 0;
      test.packageMember = 0;
      test.privateMember = 0; ✗
      }
}
```

package otherPackage

subclass access

```
import packageA.TestMemberAccessFromClass;

public class TestMemberAccessFromSubclass
       extends TestMemberAccessFromClass {

   public TestMemberAccessFromSubclass() {
      publicMember = 0;
      protectedMember = 0;
      packageMember = 0; ✗
      privateMember = 0; ✗
   }
}
```

world access

```
import packageA.TestMemberAccessFromClass;

public class TestMemberAccessFromWorld {

   TestMemberAccessFromClass test =
      new TestMemberAccessFromClass();

   public TestMemberAccessFromWorld(){
      test.publicMember = 0;
      test.protectedMember = 0; ✗
      test.packageMember = 0; ✗
      test.privateMember = 0; ✗
   }
}
```

access to ⟶	private	default (package)	protected	public
same class	✓	✓	✓	✓
same package	✗	✓	✓	✓
subclass (in same package)	✗	✓	✓	✓
subclass (in other package)	✗	✗	✓	✓
other package	✗	✗	✗	✓

instance vs class members and methods

```
public class PublicClass {

   public static int classMember;
   public int instanceMember;

   public static void classMethod(){
      classMember = 0;
      instanceMember = 0; ✗
   }

   public void instanceMethod(){
      classMember = 0;
      instanceMember = 0;
   }
}
```

```
public class TestAccess {

   public TestAccess(){
      PublicClass pc = new PublicClass();

      PublicClass.classMember = 0;
      pc.classMember = 0;                    same access

      PublicClass.instanceMember = 0; ✗      Class access to instance member
      pc.instanceMember = 0 ;

      PublicClass.classMethod();
      pc.classMethod();                      same calls

      PublicClass.instanceMethod(); ✗        Class access to instance method
      pc.instanceMethod();
   }

   public void testValues(){
      PublicClass pc1 = new PublicClass();
      PublicClass pc2 = new PublicClass();

      pc1.instanceMember = 0 ;
      pc2.instanceMember = 1 ;
      PublicClass.classMember = 2;

      System.out.println(pc1.instanceMember);   prints 0
      System.out.println(pc1.classMember);      prints 2
      System.out.println(pc2.instanceMember);   prints 1
      System.out.println(pc2.classMember);      prints 2
      }
}
```

Final vs non-final members and methods

```
public class FinalOrNot {

   public static final int FINAL_CLASS_VARIABLE;
   static{
      FINAL_CLASS_VARIABLE = 0;
   }

   public static final int UNASSIGNED_FINAL_CLASS_VARIABLE;   ✗  ← never assigned
   public static final int SET_FINAL_CLASS_VARIABLE = 0 ;

   public final int finalInstanceVariable;
   public final int setFinalInstanceVariable = 0 ;
   public final int unassignedFinalInstanceVariable;   ✗  ← never assigned

   public  FinalOrNot(int val){

      int i = FinalOrNot.FINAL_CLASS_VARIABLE;
      FinalOrNot.FINAL_CLASS_VARIABLE = val;   ✗  ← second assignment

      finalInstanceVariable = val;
      finalInstanceVariable = val;   ✗  ← second assignment
      setFinalInstanceVariable = val;   ✗  ← second assignment
   }

   public void normalMethod(){String s="some code";};

   public final void finalMethod(){String s="some code";};
}
```

```
public class TestAccess {

   public TestAccess(){
      int i = FinalOrNot.FINAL_CLASS_VARIABLE;
      FinalOrNot.FINAL_CLASS_VARIABLE = val;   ✗  ← cannot re-assign

      FinalOrNot f1 = new FinalOrNot(1);
      FinalOrNot f2 = new FinalOrNot(2);

      System.out.println(FinalOrNot.FINAL_CLASS_VARIABLE);  ← prints 0
      System.out.println(f1.finalInstanceVariable);  ← prints 1
      System.out.println(f2.finalInstanceVariable);  ← prints 2

      f1.finalInstanceVariable = val;   ✗  ← cannot re-assign
   }
}
```

```
public class FinalOrNotSubclass extends FinalOrNot {

   public FinalOrNotSubclass(){super(0);}

   public void normalMethod(){
      String s="some other code";
   }

   public final void finalMethod(){   ✗  ← cannot override "final" methods
      String s="some other code";
   }
}
```

Abstract classes and interfaces

```java
public abstract class AbstractClass {

   public int instanceVariable             = 0;
   public static int classVariable         = 0;
   final public static int CLASS_CONSTANT  = 0;

   abstract static public void abstractClassMethod(); ✗
   abstract public void abstractInstanceMethod();

   abstract public void abstractInstanceMethodWithBody(){ ✗
      String s = "some code goes here";
   }

   public void concreteInstanceMethodWithBody(){
      String s = "some code goes here";
   }

   final public void finalConcreteMethodWithBody(){
      String s = "some code goes here";
   }

   public void concreteInstanceMethod(); ✗
}
```

cannot have abstract class methods

cannot have defined abstract instance methods

cannot have undefined concrete instance methods

```java
public class ConcreteClass extends AbstractClass {

   abstract public void someAbstractMethod(); ✗

   AbstractClass abstractClassInstance = new AbstractClass(); ✗

   @Override
   public void abstractInstanceMethod() {
      System.out.println("abstractInstanceMethod must be implemented");
   }

   @Override
   public void concreteInstanceMethodWithBody(){
      System.out.println("Unnecessary but can be done");
   }

   @Override
   public void finalConcreteMethodWithBody(){ ✗
      String s = "Cannot be done";
   }
}
```

concrete class cannot have abstract methods

cannot instanciate abstract classes

cannot override "final" methods

```java
public interface MyInterface {

   //Every field declaration in the body of an interface is implicitly
   //public, static, and final. It is permitted to redundantly specify
   //any or all of these modifiers for such fields.
   // see https://docs.oracle.com/javase/specs/jls/se7/html/jls-9.html
   public static final int CONSTANT = 0;

   // Every method declaration in the body of an interface is
   // implicitly public and abstract. It is permitted to redundantly
   // specify any or all of these modifiers.
   abstract public void abstractInstanceMethod();

   abstract public void abstractMethodWithBody(){ ✗
      String s = "some code goes here";
   }

   abstract static void abstractClassMethod(); ✗
}
```

cannot have defined methods

cannot have "class" methods

```java
public class MyClass implements MyInterface {

   MyInterface interfaceInstance = new MyInterface(); ✗

   @Override
   public void abstractInstanceMethod() {
      String s = "must be implemented";
      System.out.println(MyInterface.CONSTANT);
   }
}
```

cannot instanciate interfaces

must be implemented

	concrete class	abstract class	interface
Instances can be created	✓	✗	✗
Has instance variables and methods	✓	✓	✗
Can have constants	✓	✓	✓
Can declare abstracts methods	✗	✓	✓